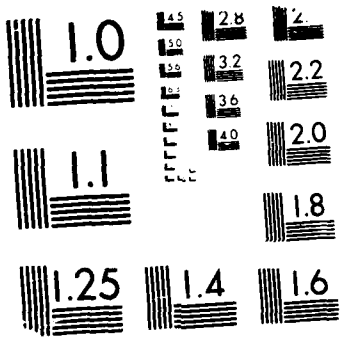
AD-A191 953 AN EXPERT SYSTEM MODEL USING PREDICATE TRANSITION NETS 1/1
(U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR
INFORMATION AND D D M PERDU ET AL JAN 88
UNCLASSIFIED LIDS-P-1736 N00014-85-K-0782 F/G 12/5 NL



JANUARY 1988 LIDS - P - 1736



# AN EXPERT SYSTEM MODEL USING PREDICATE TRANSITION NETS*

Didier M. Perdu
Alexander H. Levis**

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139
USA



## ABSTRACT

A Predicate Transition Net model of expert systems with fuzzy logic is proposed as a means for dealing with uncertainty. Attention is focused on consultant expert systems which use the production rules formalism to represent knowledge. Models of the basic fuzzy logical operators are presented base on the Predicate Transition Nets formalism. The combination of these operators allows to represent the links among the rules of a knowledge base and to simulate dynamically the behavior of an expert system in its search for a solution to a problem in its domain of expertise. Finally, two applications of this model are described: assessment of parallelism in a knowledge base and the evaluation of time-related measures for expert systems.



* This work was conducted at the MIT Laboratory for Information and Decision Systems with support provided by the Basic Research Group of the Technical Panel on $C^3$ of the Joint Directors of Laboratories through the Office of Naval Research under Contract no. N00014-85-K-0782.

** Address all correspondence regarding this paper to Dr. Alexander H. Levis, MIT, Room 35-410/LIDS, Cambridge, MA 02139, USA.

## 1.0 INTRODUCTION

Knowledge base expert systems show properties of synchronicity and concurrency which makes them suitable for being represented with the Predicate Transition Net formalism. The rules of a knowledge base have to be checked in a specific order depending on the strategy used to solve the problem and on the current facts deduced so far by the system in the execution of previous rules. This paper proposes a model of an expert system using production rules to represent knowledge. Fuzzy logic is used to deal with uncertainty and Predicate Transition Nets are used to represent the basic fuzzy logical operators AND, OR and NOT that appear in this kind of rules. An extension of the standard inference net formalism is obtained by the combination of these operators which permits to represent the dynamical behavior of an expert system. The determination of the simple paths and of the slices of the net leads to the identification of the rules which can be checked concurrently by the inference engine in a parallel architecture computer. This net allows also the identification of the rules scanned by the system to produce an answer to a specific problem and to deduce its response time depending on the number of rules scanned and on the number of interactions with the user.

## 2.0 STRUCTURE OF THE EXPERT SYSTEM

Knowledge Based Expert Systems, commonly called Expert Systems, are - in theory - able to reason using an approach similar to the one followed by an expert when he solves a problem within his field of expertise. An expert system can be used for many purposes : to control, to diagnose, to solve problems, to plan, to design, ....

This paper proposes a model for the most common kind of expert system : the **consultant expert system**, as described by Johnson and Keravnov (1985). Most systems engage in a dialogue with the user, the computer acting as a "consultant," by suggesting options on the basis of its knowledge and the symbolic data supplied by the user. The dialogue terminates when a decision or a recommendation is reached. The formalism used to represent knowledge in consultant expert systems is the production system model.

Moving from known items of information to unknown information is the vital process of a consultant system. The user of a consultant expert system has "observed" some particular state of affairs within the domain of the system's expertise and submits these observations to the system. Examples of these states are a sick person, a faulty machine or a malfunctioning business

environment. Based on the observations, the system makes inferences and suggests new routes of investigation which will yield high grade information. Interactions continue until the system finds the most likely explanation of the observations.

There are three distinct components in an expert system, the Knowledge Base, the Fact base, and the Inference Engine.

## 2.1 The Knowledge Base

The knowledge base contains the set of information specific to the field of expertise. Knowledge is expressed in a language defined by the expert. The knowledge base is a collection of general facts, empirical rules, and causal models of the problem domain. A number of formalisms exist to represent knowledge. The most widely used is the production system model in which the knowledge is encoded in the form of antecedent-consequent pairs or IF-THEN rules. A production rule is divided in two parts :

- A set of conditions (called left-hand side of the rule) combined logically together with a AND or a OR operator,

- A set of consequences or actions (called also right-hand side of the rule), the value of which is computed according to the conditions of the rule. These consequences can be the conditions for other rules. The logical combination of the conditions on the left- hand side of the rule has to be true in order to validate the consequences and the actions.

An example of a production rule is :

IF the flying object has delta wings AND
the object flies at great speed
THEN the flying object is a fighter plane.

The conditions "the flying object has delta wings" and "the object flies at a great speed" have to be true to attribute the value true the consequence "the flying object is a fighter plane."

*Inference Net.* The relationships among the rules of a production system can be represented with an inference net. The net shows graphically the logical articulation of different facts or subgoals,

and identifies which rules are used to reach a specific goal. Let us consider the following production rules :

if A AND B, then C
if D OR E, then F
if NOT G, then H.

These rules are represented in the inference net formalism on Figure 1.

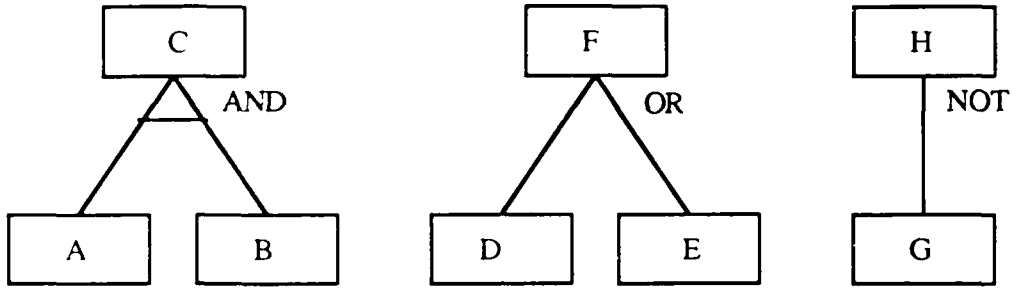



Figure 1 Representation of the logical operators in the inference net formalism

The Predicate Transition Net model developed in this paper is an extension of the inference net formalism and permits the explicit representation of the rules of a knowledge base and the relationships among them.

## 2.2 The Fact Base.

The fact base, also known as context or working memory, contains the data for the specific problem to be solved. It is a workspace for the problem constructed by the inference mechanism from the information provided by the user and the knowledge base. The working memory contains a trace of every line of reasoning previously used by memorizing all the intermediate results. Therefore, this can be used to explain the origin of the information deduced or to describe the behavior of the system.

## 2.3 The Inference Engine.

The Inference Engine is used to monitor the execution of the program by using the knowledge base to modify the context. It uses the knowledge and the heuristics contained in the knowledge base to solve the problem specified by the data contained in the fact base. In the production system modeled in this paper, the rules are of the kind, A -> B, saying that if A is valid, B can be deduced. The inference engine selects, validates and triggers some of these rules to reach the solution of the problem.

*Logic used* : In order to deal with uncertainty in items of evidence, fuzzy logic (Zadeh, 1965, 1983) has been implemented in the model to combine logically the conditions of the left-hand side of the production rules. The value of a rule or a fact is either unknown or a number, pi, between 0 and 1 representing the degree of truth associated with it. The operators AND, OR, and NOT execute operations on these degrees of truth as follows:

$$p1 \text{ AND } p2 = \min(p1,p2)$$
$$p1 \text{ OR } p2 = \max(p1,p2)$$
$$\text{NOT } p1 = 1-p1.$$

*Process of Selection and Firing of Rules* : Among the strategies used by the inference engine to select the rules, forward chaining and backward chaining are the most common. In **forward chaining**, the inference mechanism works from an initial state state of known facts to a goal state. It finds first all the rules that match the context, then it selects one rule based on some conflict resolution strategy, and then execute the selected rule. Facts are inputs to the system. The most appropriate hypothesis that fits the facts is deduced. For **backward chaining**, the system tries to support a hypothesis by checking known facts in the context. If these known facts do not support the hypothesis, the preconditions needed for the hypothesis are set up as subgoals. The process for finding a solution is to search from the goal to the initial state, it involves a depth-first search.

In order to simulate the behavior of an expert system, the process of selection and firing of rules done by the inference engine has been modeled when a backward chaining strategy is used. A trigger is associated with every rule (or operator). A rule is selected by the inference engine when the trigger is activated. Only one rule at a time can be activated and the continuation of the selection and firing process is done according to the result of the rule :

- If the result is unknown, the rule is put in memory and the rule which gives the value of the first unknown precondition is selected.

- If the result is known, the last rule which was put in memory is selected again because the produced result is the value of one of its preconditions.

Let us consider the example where we have two rules :

$$B \Rightarrow C \qquad (1)$$
$$A \Rightarrow B \qquad (2)$$

and where the degree of truth of the fact A is known.

The inference engine selects first rule (1). The degree of truth of C is unknown because the degree of truth of B is unknown. Rule (1) is then de-activated and put in memory. Then rule (2) is selected. Since the value of A is known, the value of B is deduced. Rule (1), which is the last to have been put in memory, is selected again and the answer C is obtained.

*Search for efficiency* : The process of selection and firing of rules described above is repeated by recursion until the final answer is found ; the process can last a long time. In the search for efficiency and performance, unnecessary computations must be avoided. In some cases, there is no need to know the values of all the preconditions of a rule to deduce the value of its consequence. For example, in boolean logic, if we have the rule :

$$A \text{ AND } B \Rightarrow C.$$

and we know that :

$$A \text{ is false,}$$

then the consequence C is false and there is no need to look for the value of B to conclude that ; the set of rules giving the value of B can be pruned.

In systems using fuzzy logic, this avoidance of unnecessary computations is all the more important as computations are more costly in time and memory storage than in systems using boolean logic. The problem is that little improvement in performance is obtained, if extra

computation is avoided only in the case of complete truth (for the operator OR) or of complete falsity (for the operator AND). The solution lies in the setting of thresholds for certain truth and certain falsity. For example, in the case of the operator AND, if we have

A AND B => C

and if we know that the degree of truth of A is less than the threshold of certain falsity, then we can deduce that the degree of truth of the consequence C is less than the degree of truth of A and, therefore, less than the threshold of certain falsity. There is no need to know the degree of truth of the precondition B. The thresholds for which no further search is required in the execution of the operators are set to 0.8 for certain truth in the operator OR and 0.2 for certain falsity in the operator AND. A rule or fact having a degree of truth larger or equal to 0.8 (resp. less or equal to 0.2) will be considered to be true (resp. false). Therefore, the logic takes into account the unknown rules or facts.

## 3.0 CHARACTERISTICS OF THE PREDICATE TRANSITION NETS USED IN THE MODEL

Predicate Transition Nets have been introduced by Genrich and Lautenbach (1981) as an extension of the ordinary Petri Nets (Peterson, 1980; Reisig, 1985) to allow the handling of different classes of tokens.The Predicate Transition Nets used in the model have the following characteristics.

### 3.1 Tokens

Each **token** traveling through the net has an identity and is considered to be an individual of a given class called variable. Each variable can receive different names. For this model, two classes of tokens are differentiated :

(1) The first class, denoted by P, is the set of the real numbers between 0 and 1, representing the degrees of truth of the facts or items of evidence. The names of the individual tokens of these classes will be p, p1, p2.

(2) The second class is denoted by S. The individuals of this class can only take one

value. Only one token of this class will travel through the net and will represent the action of the inference engine in triggering the different rules.

## 3.2 Places

**Places** are entities which can contain tokens before and after the firing of transitions. Three kinds of places are differentiated :

(1) places representing a fact or the result of a rule and containing tokens of the class P or no token at all,

(2) places used by the system as triggers of operators and containing the token of the class S. These places and the connectors connected to these places are represented in bold style in the Figures and constitute the **system net**.

(3) places allowed to contain different kinds of tokens (P and S) and which are used to collect the tokens necessary for the enabling of the transitions of which they are the input places.

The **marking** of a place is a formal sum of the individual tokens contained in the place. For example, a place A containing a token of the class P, p1 and the token of the class S has the marking M(A) :

$$M(A) = p1 + S$$

## 3.3 Connectors and Labels

Each **connector** has a label associated with it which indicates the kinds of tokens it can carry. A special grammar is used on the labels to define in what way tokens can be carried. The labels of connectors linking places to transitions contain conditions tthat must be fulfilled for them to carry the tokens. The labels of connectors linking transitions to places indicate what kind of token twill appear in the places after the firing of the transition.

The following notation in labels is used :

When token names are joined by the sign "+" then the tokens defined by these names have to be carried at the same time. For example, the label "p + S" indicates that one token of the class P and one token of the class S have to be carried together at the same time by the connector.

When token names are joined by the sign "," then the tokens defined by these names can be carried at different times but not together. For example, the label "p, S" indicates that either a token of the class P or a token of the class S can be carried.

Mixing of notation is possible. The label "p+S, S" indicates that the connector can carry either a token of the class P and a token of the class S or only one token of the class S.

A connector without label has no constraint on the kind of tokens it can carry.

In some cases, the connector has to carry the token of class S when there is no token of the class P involved in the firing of a transition. The statement "absence of token of the class P" is denoted by the symbol Ø. This symbol is used in the labels, as if it was a class of tokens, in association with the names of the other classes. The symbol Ø is used in the following cases :

(1) The label "S+Ø" means that the connector can carry a token of the class S, if there is no token of the class P.

(2) The label "(S+p), (S+Ø)" means that the connector can carry either a token of the class S and a token of the class P, or a token of the class S, if there is no token of the class P.

## 3.4 Transitions

Transitions have attached to them a predicate which is a logical formula (or an algorithm) built from the operations and relations on variables and tokens in the labels of the input connectors. The value (true or false) taken by the predicate of a transition depends on the tokens contained in the input places of the transition. When the predicate has the value "true", the transition is enabled and can fire. In the model of the consultant expert system, predicates are conditions on tokens of the class P.

A transition without predicates is enabled as soon as all the input places contain the tokens

specified by the labels of the connectors.

Transitions with predicates are represented graphically with squares or rectangles. The predicate is written inside. Transitions without predicates are represented with bars as in ordinary Petri Nets.

### 3.5 Firing Process

The conditions of enabling of a transition are: (1) the input places contain the combination of tokens specified by the labels of the connectors, and (2) the predicate of the transition is true. If these two conditions are fulfilled, the transition can fire. In the firing process, tokens specified by the input connectors are withdrawn from the corresponding input places and tokens specified by the output connectors are put in the output places.

Let us consider the example shown on Figure 2 :

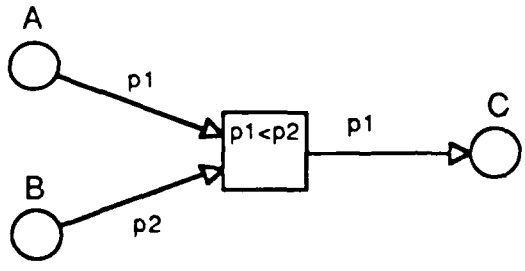



Figure 2 Example of a transition with a predicate

The condition "p1 < p2" written in the transition represented by a square is true when the value of the token named p1 coming from place A is less than the value of the token named p2 coming from place B, as specified by the connectors. In this case, the transition is enabled and can fire; the tokens p1 and p2 are withdrawn from the places A and B and a token p1 is put in place C.

## 4.0 LOGICAL OPERATOR MODELS

In order to construct the model of the expert system using Predicate Transition Nets, it is

necessary to construct first models of the logical operators AND, OR, and NOT. The results are shown in Figures 3, 4 and 5. Let us describe now what happens in the operator AND (the operators OR and NOT behave in a similar way).

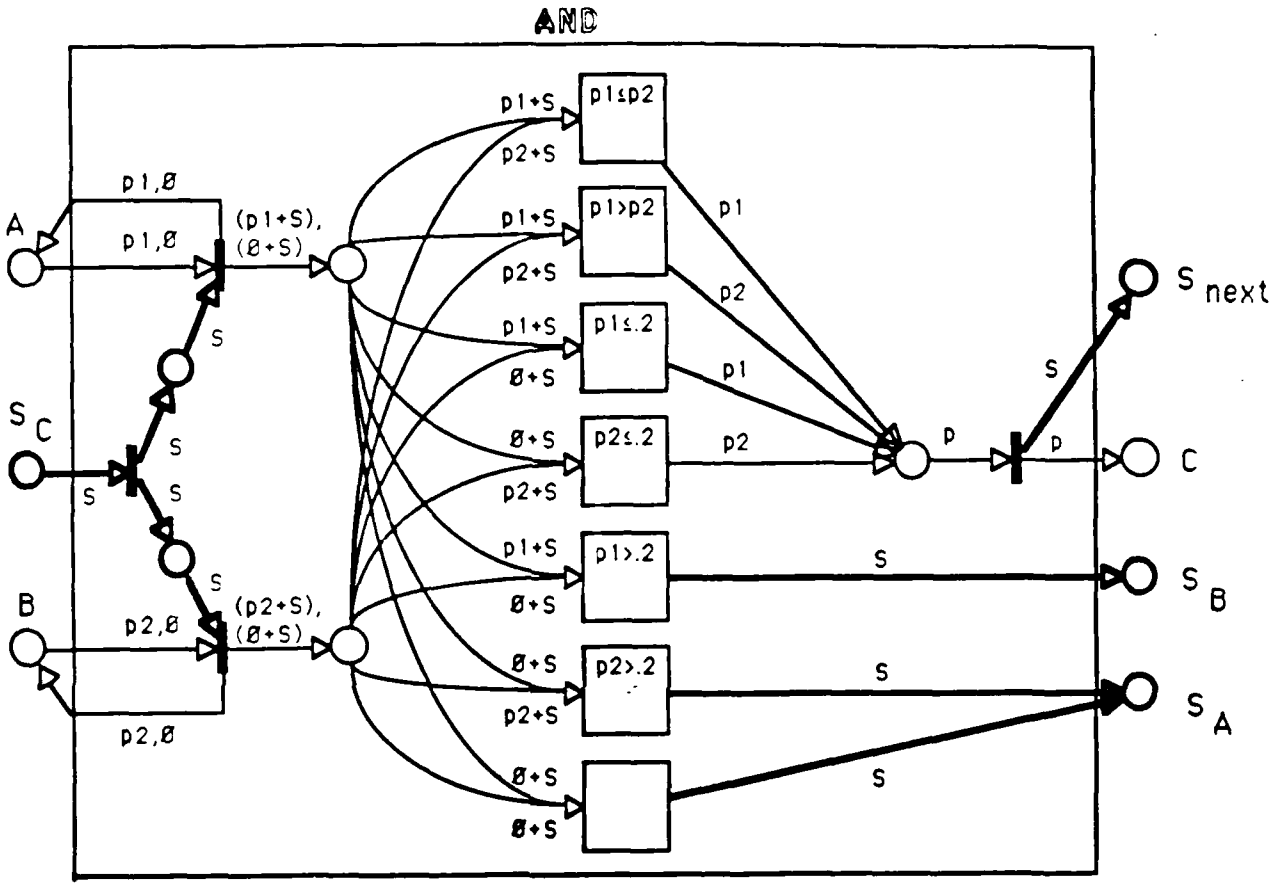



Figure 3 Model of the operator AND

The operator drawn in Figure 3 realizes the operation :

A AND B => C.

It can be represented as a black box, having three inputs : A, B and $S_C$ (the trigger) and six outputs : C (the result), A, B (memorizing of the input value) and three system places $S_A$, $S_B$ and $S_{next}$. Only one of those system places (represented in bold style in the figures) can have a system token at the output. $S_{next}$ will contain a system token, if the result of the operation is known, i.e., if C contains a token of the class P. This shows that the next operation can be

performed. If the result is unknown, i.e., the two inputs are not sufficient to yield a result, the system token is assigned to $S_A$ or $S_B$ in order to get the values of these unknown inputs. A system token will be assigned to $S_A$ if (i) C is unknown and (ii) A is unknown or if A and B are both unknown. The system token will be assigned to $S_B$ if C is unknown and only B is unknown.

The execution of the operation will start only if there is a system token in $S_C$. We denote by $S_C$ the trigger place of the operator computing C. As soon as there is a token in $S_C$, the two input transitions are triggered by the allocation of a system token (S) at the input places of these transitions. The values of A and B are therefore reproduced in A and B and in the output place of each of the transitions. These places contain also a system token, which will ensure the enabling of the following transition (i.e., that the two inputs are present). These two places are the input places of seven different transitions which have disjoint conditions of enabling. Only one of

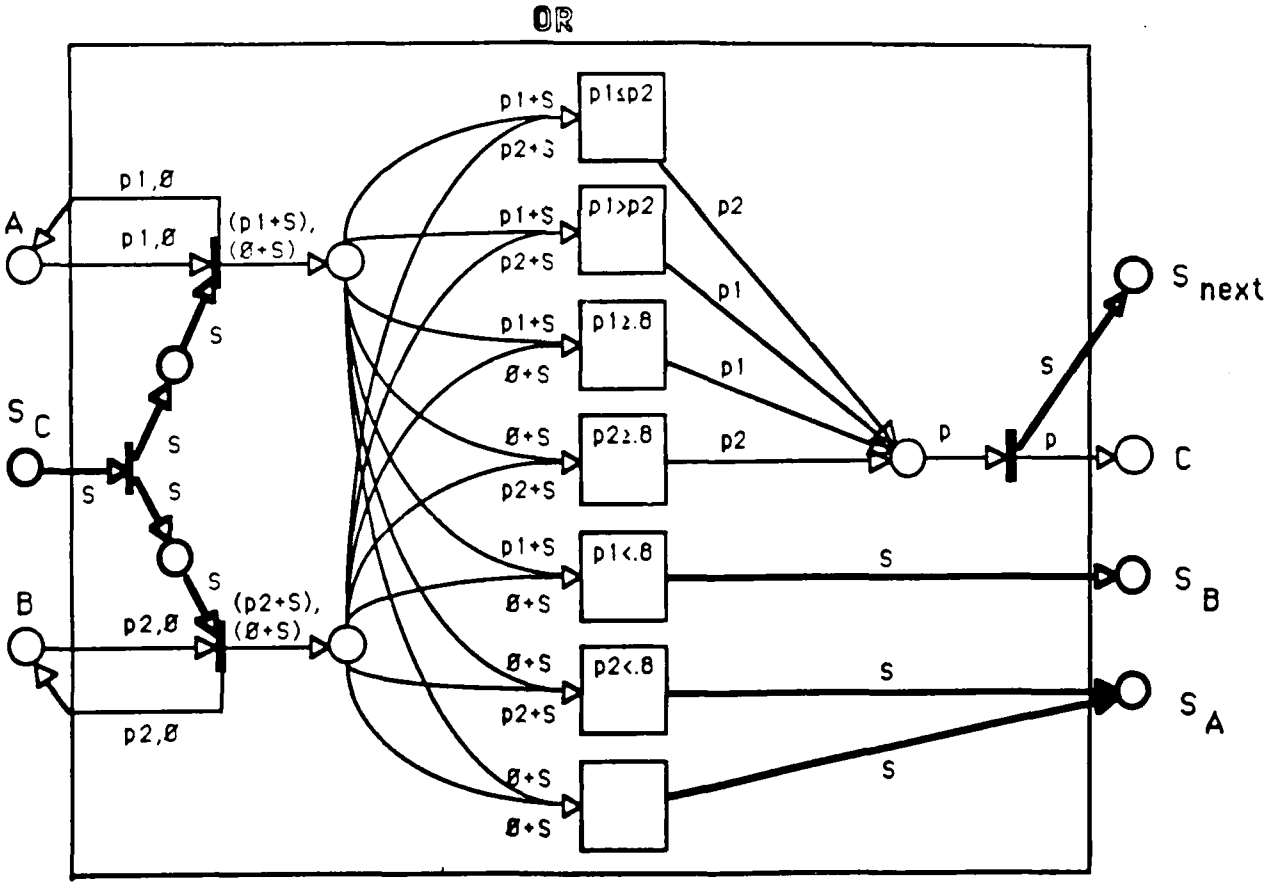



Figure 4 Model of the operator OR

these transitions can be enabled and can fire. At the firing, the result, if any, is given in the result place and then in C, while the system token is assigned either to $S_{next}$, or to $S_A$, or to $S_B$.

These operators can be compounded in super-transitions. The model can be generalized to operators with more than two inputs by combining these basic operators.

An example of the use of these logical operators is shown on the next section, where a simple inference net is modeled and the search process in this net is simulated.

## 5.0 DYNAMIC REPRESENTATION OF AN INFERENCE NET

The connection of the super-transitions representing the logic operators to places representing the items of evidence leads to a dynamic representation of an inference net. It allows to show explicitly how the inference engine scans the knowledge base. By running a simulation program, we can see in real time what the steps of reasoning are, the possible deadlocks, or mistakes. It allows one to identify the parts of the knowledge base where the knowledge representation is incorrect.

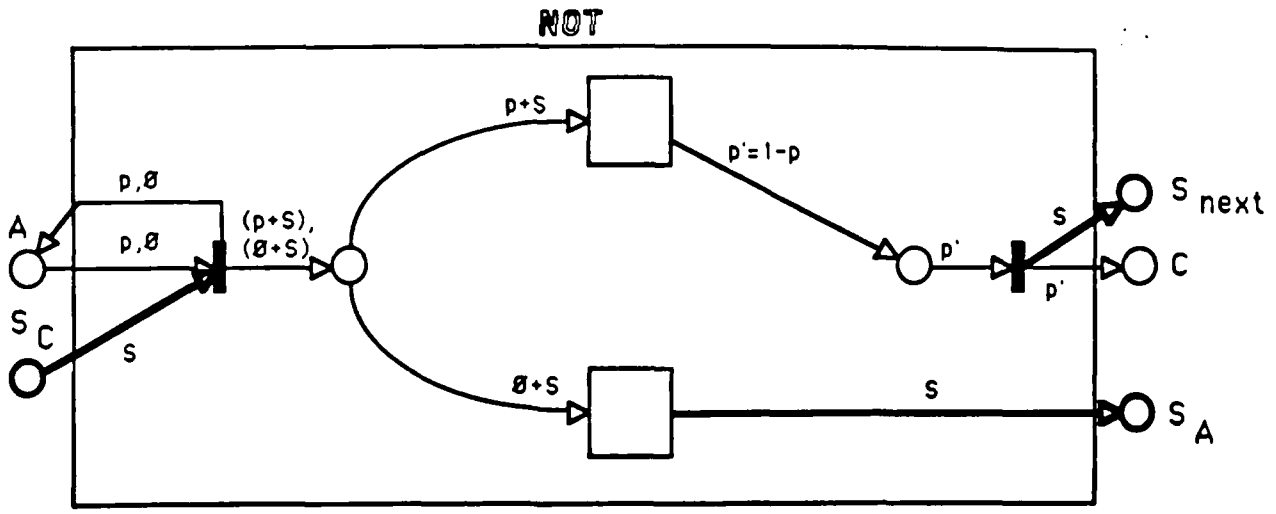



Figure 5 Model of the operator NOT

Let us consider the simple symbolic system containing the following rules :

if A and B => E
if C and D => D
if E or F => G

The standard representation of the inference net of this system (see section 3.1) is shown in Figure 6.

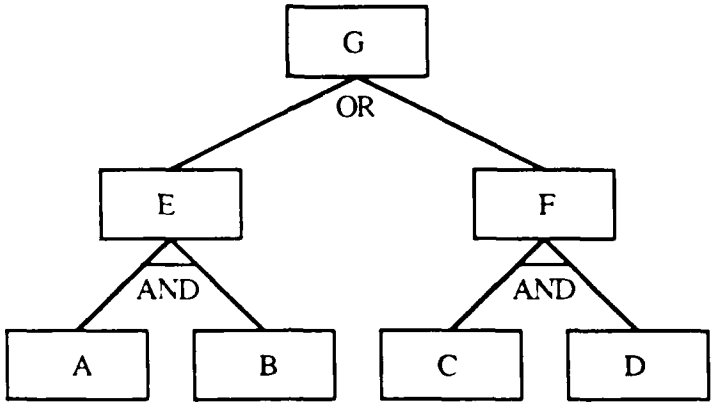



Figure 6 Standard representation of the inference net of the example.

The representation of the inference net with Predicate Transition Net is deduced from this representation by :

(1) replacing the rectangles representing the subgoals with the places of our model.

(2) replacing the formalism AND, OR, and NOT by the models of the operators aggregated in super-transitions, and linking these places to those transitions (including the self loops).

(3) linking the system places of each operator according to the rules described in section 4 for the scheduling of the checking of the unknown subgoals.

The representation of the inference net of the simple symbolic system, using the Predicate Transition Net models of the logic operators, is shown on Figure 7. The interface module with the user has been added through the places IA, IB, IC and ID, where the user can enter the

degrees of truth of A, B, C and D.

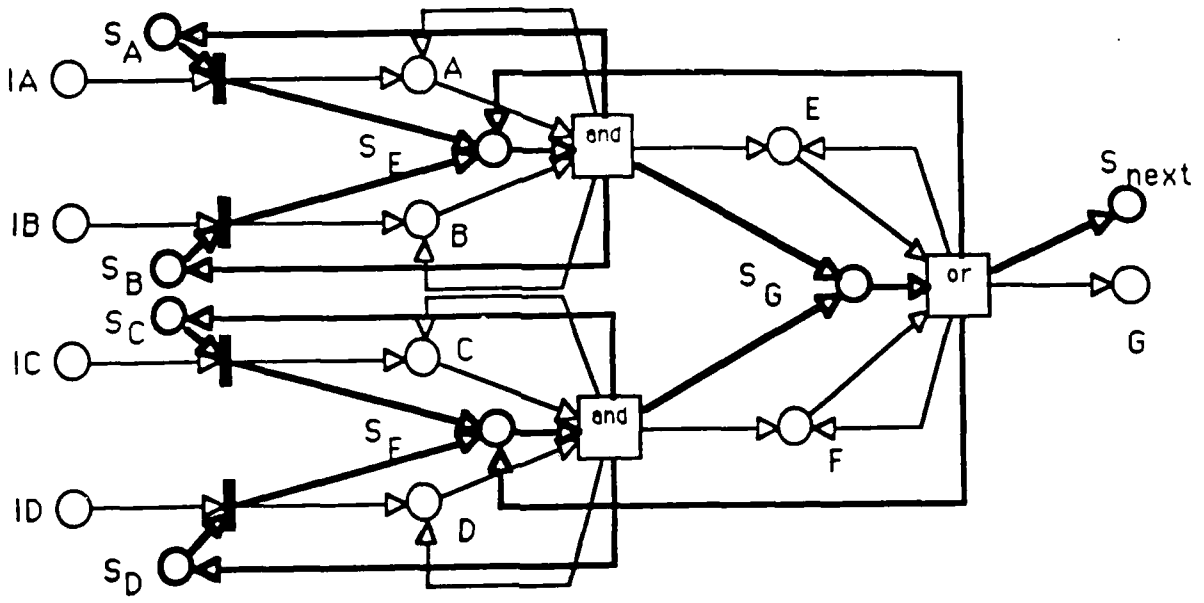



Figure 7 Inference net of a simple symbolic system, using the Predicate Transition Nets formalism

The simulation of the propagation of the tokens in this net allows one to observe the reasoning process followed by the system. The mapping of the different places of the net at each step of the process of the simulation is shown on Table 1.

The search for the degree of truth of the goal G starts when the system token is put in the system place $S_G$, at the beginning of the search (step 1). The degree of truth of G cannot be evaluated when the operator OR is executed. The system token is therefore assigned to $S_E$ for the checking of the subgoal E (step 2). The execution of the operator AND cannot lead to a result for E and the system token is allocated to $S_A$ (step3), which triggers an interaction session with the user to get the degree of truth of A. The user enters this value (say 0.9) through IA (step 4) which is assigned to A, while the system token is assigned to $S_E$ (step 5). Since, the degree of truth of A is larger than 0.2, the result of the operator AND cannot be given in E and the system token is assigned to $S_B$ (step 6) to get the degree of truth of B (say 0.8) through IB (step 7). The system token is then re-assigned to $S_E$ to trigger the operator AND (step 8), which can now be executed. The minimum of the degrees of truth of A and B, 0.8, is put in E, while the system token is assigned to $S_G$ (step 9). Since the degree of truth of E is equal to 0.8, the operation OR can be

performed to produce the result G equal to 0.8. The system token is allocated in $S_{next}$ (step 10). The subgoal F has not been checked and all the part of the net which is used to evaluate F has been pruned.

TABLE 1 Mapping of the Places at the different steps of the simulation

| | A | IA | B | IB | C | IC | D | ID | E | F | G | $S_A$ | $S_B$ | $S_C$ | $S_D$ | $S_E$ | $S_F$ | $S_G$ | $S_{next}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Step 1 | | | | | | | | | | | | | | | | | | S | |
| Step 2 | | | | | | | | | | | | | | | | S | | | |
| Step 3 | | | | | | | | | | | | S | | | | | | | |
| Step 4 | | 0.9 | | | | | | | | | | S | | | | | | | |
| Step 5 | 0.9 | | | | | | | | | | | | | | | S | | | |
| Step 6 | 0.9 | | | | | | | | | | | | S | | | | | | |
| Step 7 | 0.9 | | | 0.8 | | | | | | | | | S | | | | | | |
| Step 8 | 0.9 | | 0.8 | | | | | | | | | | | | | S | | | |
| Step 9 | 0.9 | | 0.8 | | | | | | 0.8 | | | | | | | | | S | |
| Step 10 | 0.9 | | 0.8 | | | | | | 0.8 | | 0.8 | | | | | | | | S |

## 6.0 SOME APPLICATIONS OF THE MODEL

### 6.1 Assessment of Parallelism

The search for a solution with an expert system is very costly in time and memory storage and some limits exist regarding the size and the kind of problem that can be solved. A way to improve the performance is to dispatch the problem to different processors, each of them solving a part of the problem concurrently. Results of each of the parts are sent to the appropriate processors through a message passing protocol. The problem is that message passing is costly in time. Therefore, the different tasks allocated to each processor have to be chosen very carefully in order to minimize (i) the number of communications among the processors and (ii) the average idle time in processors waiting for the result of a computation done on another processor. The most reasonable way to do this is to allocate to each processor a part of the problem which is as decoupled as possible from the other parts of the computation. The Predicate Transition Net

model of the inference net allows to schedule the allocation of the rules to different processors. This is done in the following way :

We first transform the standard representation of the inference net of the problem to be solved into its representation with Predicate Transition Nets, described in section 5. This representation is then modified by first suppressing the system net representing the action of the inference mechanism to select the rules (i.e. all the places and connectors represented in bold style on the Figures), and second, by suppressing the self loops representing the memorizing in the fact base of the intermediate results.

For example, the transformed net obtained from the representation of the example of the simple symbolic system, obtained from its representation shown on Figure 7, is shown on Figure 8.

The obtained net looks like an ordinary Petri Nets and the simple paths and the slices (Hillion, 1986; Jin et al., 1986) of the transformed net can be determined. The slices indicate the operations which can be performed concurrently, while the simple paths indicate the sets of the dependent rules which have to be scanned sequentially by the system.

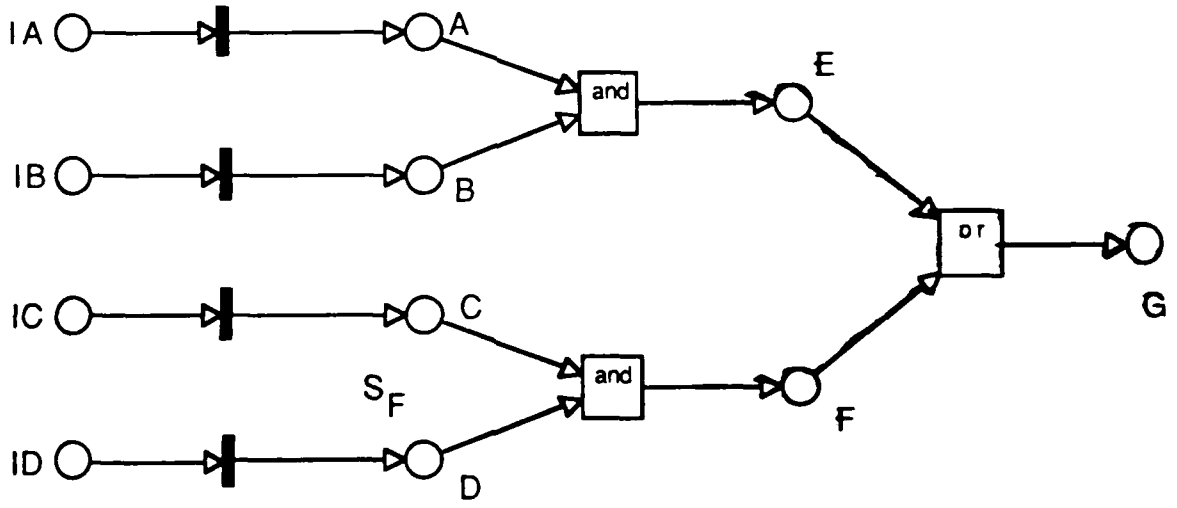



Figure 8 Example of transformed net for the assessment of parallelism.

For the simple example presented in Figure 7, we consider four processors to handle the problem. The slices of the transformed net are :

Slice 1 : { IA, IB, IC, ID}
Slice 2 : {A, B, C, D}
Slice 3 : {E, F}
Slice 4 : {G}

The simple paths are :

Simple path 1 : IA,A,E,G
Simple path 2 : IB,B,E,G
Simple path 3 : IC,C,F,G
Simple path 4 : ID,D,F,G.

An efficient way to allocate the rules to the processors is :

processor 1 : IA,A,E,G
processor 2 : IB,B (send B to processor 1)
processor 3 : IC,C,F (send F to processor 1)
processor 4 : ID,D (send D to processor 3)

This allocation requires only three message passings among the four processors.

## 6.2 Timeliness

The model allows the evaluation of the time needed to produce an ouput ; this is then used to compute the timeliness of an organization using an expert system.

The timeliness of an expert system is related to the number of rules in the rule base scanned by the system to give an answer to a specific problem or goal, and to the number of interactions with the user. The model we have defined allows a quick identification of the parts of the rule base which have been scanned, given a certain set of inputs, to reach a specific goal, since each place contains the token symbolizing the value of the rule or fact it represents.

Let us consider an expert system being used to give a certain answer in a certain environment. We represent the input $X_i$ to the system as a n-tuplet where n is the total number of questions which can be asked by the system. The answer to the questions are contained in this n-tuplet at the location corresponding to the question asked (this may not be listed in the order of appearance in time). The locations for the unasked questions are left empty. We denote by $n_i$ the number of questions asked by the system. The number of $X_i$'s might be very large but it is bounded. Given a certain environment, we can define a distribution $p_i(X_i)$ for the occurrence of the input $X_i$.

For a specific input $X_i$, we can identify $N_i$, the number of places scanned by the system to reach its goal, since they still contain the degrees of truth of the subgoals they represent. If $\tau$ is the average time to check a rule and t is the average time taken by a user to answer a question asked by the system, then the time $t_i$ to get an answer given an input $X_i$ will be :

$$t_i = N_i \tau + n_i t$$

Therefore, the average time of use T of the expert system for the set of inputs $X_i$ will be given by:

$$T = E[t_i] = \sum_i p_i t_i = \sum_i p_i N_i \tau + \sum_i p_i n_i t$$

which leads to :

$$T = E[N_i] \tau + E[n_i] t$$

where E[X] denotes the expected value of the variable X.

The time T obtained is the average time needed to get an answer from the expert system.

## 7.0 CONCLUSION

In this paper, a model of an expert system with fuzzy logic as a means for dealing with uncertainty has been developed using the Predicate Transition Nets formalism. This has been done through the modeling of the basic logic operators AND, OR and NOT. The combination of these operators makes it possible to represent the inference net of a consultant expert system using production rules and to study its behavior dynamically. Two possible applications have then been described: first, a method to assess the possible concurrency for the scanning of a rule

base by a system having a parallel architecture. Second, a method to make time-related measures of an expert system, taking into account the portion of the rule base scanned by the system and the number of interactions with the user.

REFERENCES


Genrich, H.J., and K. Lautenbach, 1981, "System Modeling with High Level Petri Nets," *Theoretical Computer Science, 13* , pp. 109-136.

Hillion, H., and A. H. Levis, 1987, "Performance Analysis of Organization using Timed Petri Nets," *Proc. 8th European Workshop on Petri Nets*, Zaragoza, Spain

Jin, V. Y., A. H. Levis, and P. Remy, 1986, "Delays in Acyclical Distributed Decisionmaking Organizations," *Proc. IFAC Symposium on Large Scale Systems : Theory and Applications*, Zurich, Switzerland.

Johnson, L., and E. T.Keravnov, 1985, *Expert Systems Technology, A Guide*, Abacus Press, Tunbridge Wells.

Peterson, J. L., 1980, *Petri Net Theory and the Modeling of Systems* , Prentice Hall, Inc., Englewood Cliffs, NJ.

Reisig, W., 1985, *Petri Nets, An Introduction*, Springer Verlag, Berlin.

Zadeh, L. A., 1965, "Fuzzy Sets," *Information and Control* , Vol. 8, pp. 338-353.

Zadeh, L. A., 1983, "The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems," *Fuzzy Sets and Systems*, Vol. 11, pp. 199-227.

END

DATE

FILMED

5—88

DTIC